第三部分：时序逻辑电路设计

一. 锁存器和触发器

二. 有限状态机

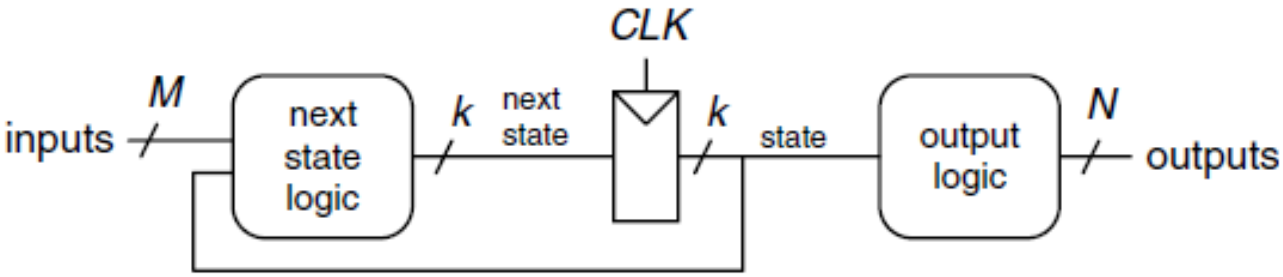
1. Moore型有限状态机
2. Mealy型有限状态机

三. 时序逻辑电路设计分析

时序电路特点

❖时序电路实例分析

- 存在直通环路，输出直接反馈到输入。这类时序电路存在不良竞争，门电路延时、温度、电压等都可能影响功能，且难发现。
- 改进措施：（1）在环路插入寄存器以断开环路，电路变成组合逻辑电路和寄存器的组合；（2）加入时钟，寄存器只在时钟边沿达到时发生改变，时钟周期保证在下一个时钟沿达到之前，输入到寄存器的信号都稳定下来——同步时序电路。



❖同步时序电路特点

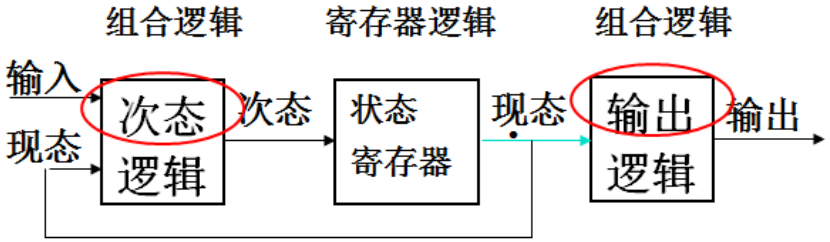
- 每个电路元件都是组合逻辑或寄存器，且至少有一个寄存器；
- 每一个环路至少有一个寄存器；
- 所有寄存器接收同一个时钟信号。
- 同步时序电路可以描绘成有限状态机。

有限状态机概述

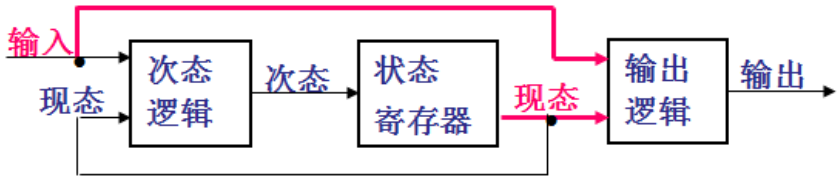
- 有限状态机（**Finite State Machine，FSM**）是表示有限个状态以及这些状态之间的转移和动作等行为的离散数学模型。
- FSM常用于时序逻辑电路设计，尤其适于设计数字系统的控制模块。具有速度快、结构简单、可靠性高、逻辑清晰、复杂问题简单化的优点。

有限状态机的分类

- 有限状态机是组合逻辑和寄存器逻辑的特殊组合。组合逻辑部分包括次态逻辑和输出逻辑，分别用于状态译码和产生输出信号；寄存器逻辑部分用于存储状态。
- 根据输出信号产生的机理不同，状态机可以分成两类：
  - 摩尔(Moore)型状态机--输出信号仅与当前状态有关
  - 米里(Mealy)型状态机--输出信号与当前状态及输入信号有关



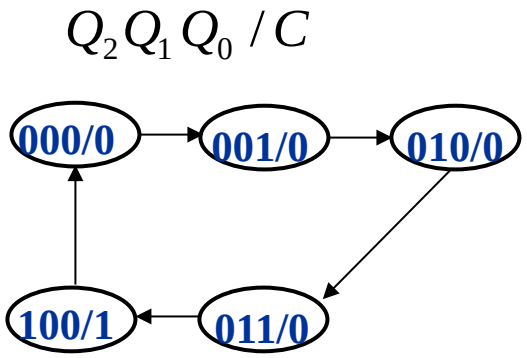
Moore型状态机典型结构



Mealy型状态机的典型结构

有限状态机的表示方法

- 状态机有2种表示方法
  - 状态图（State Diagram）、状态表（State Table）
  - 二者可以相互转换



状态（转换）图

状态（转换）表

$Q_2^nQ_1^nQ_0^n$	$Q_2^{n+1}Q_1^{n+1}Q_0^{n+1}$	C
0 0 0	0 0 1	0
0 0 1	0 1 0	0
0 1 0	0 1 1	0
0 1 1	1 0 0	0
1 0 0	0 0 0	1

进一步理解状态机

- 什么是状态？
  - 把寄存器s的值作为编码（这些寄存器也被称为状态寄存器）
  - 每个编码对应一个状态
- 状态寄存器：就是寄存器
  - 寄存器：每个时钟上升沿时存储输入端的值
- 状态寄存器的值：取决于次态逻辑的输出（即寄存器的输入）
- 次态逻辑：根据输入和当前状态编码值，计算下一个状态编码值
  - 就是组合逻辑。命名上与输出逻辑区分开
  - 决定了状态间如何转移！
- 输出逻辑：根据状态及输入，计算该状态下输出取值
  - Moore：组合逻辑的输入仅有状态
  - Meely：组合逻辑的输入涉及输入

进一步理解状态机

- 状态机的设计要点
  - 1、状态规划：根据设计需求决定状态数量（进而决定寄存器个数）
  - 2、状态转移：根据输入条件决定从当前状态迁移至哪个状态
  - 3、次态逻辑：根据{输入条件、当前状态编码、下个状态编码}构造每个寄存器的输入组合逻辑的真值表
    - 有了真值表，就可以得到相应的表达式
    - 有了表达式，就可以结合离散数学知识化简
  - 4、输出逻辑：根据设计需求决定在当前状态输出应该取何值
- 前述设计要点分析：
  - 要点1：决定了寄存器数量，不对正确性产生决定性影响
  - 要点2：最重要的设计环节，决定了正确性
  - 要点3和4：比较工程化的环节

有限状态机的设计方法

- 实用的状态机一般都设计为同步时序逻辑电路，它在同一个时钟信号的触发下，完成各状态之间的转移。
- 状态机设计步骤：

  1. 分析设计要求，列出全部可能状态；
  2. 画出状态转移图；
  3. 用Verilog HDL语言描述状态机，主要采用always块语句，完成3项任务：

  （1）定义起始状态（敏感信号为时钟和复位信号）；

  （2）用case或if-else语句描述出状态的转移（根据现态和输入产生次态）；

  （3）用case或if-else语句描述状态机的输出信号（敏感信号为现态）。

状态机的设计要点

- 起始状态的选择

  起始状态指电路复位后所处的状态，选择一个合理的起始状态将使整个系统简捷高效。有限状态机必须有时钟信号和复位信号。

- 状态编码方式
  - 二进制编码：采用$\log_2 N$个触发器来表示这N个状态——节省逻辑资源，但可能产生毛刺
  - 格雷编码：采用$\log_2 N$个触发器来表示这N个状态，同时相邻状态只有一个比特位不同。节省逻辑资源；又避免产生毛刺——在状态的顺序转换中，相邻状态每次只有一个比特位产生变化
  - 一位热码状态机编码（One-Hot State Machine Encoding）：采用N个触发器来表示这N个状态。可以避免状态机产生错误的输出，并且有时可简化输出逻辑。

对8个状态三种编码方式的对比

状态	二进制编码	格雷编码	一位热码编码
state0	000	000	00000001
state1	001	001	00000010
state2	010	011	00000100
state3	011	010	00001000
state4	100	110	00010000
state5	101	111	00100000
state6	110	101	01000000
state7	111	100	10000000

- 采用一位热码编码，虽然使用触发器较多，但可以有效节省和简化组合逻辑电路。
- FPGA有丰富的寄存器资源，门逻辑相对缺乏，采用一位热码编码可以有效提高电路的速度和可靠性，也有利于提高器件资源的利用率。

状态编码的HDL定义

- 状态编码的定义有两种方式：parameter和｀define语句

【例1】为state0, state1 ,state2 ,state3四个状态定义码字为：00，01，11，10

✓方式一：用parameter参数定义

用n个parameter常量表示n个状态

```
parameter   state0=2｀b00 ,
            state1=2｀b01,
            state2=2｀b11,
            state3=2｀b10;
……
    case (state)
      state0:……;
      state1:……;
      ……
```

✓方式二：用｀define语句定义

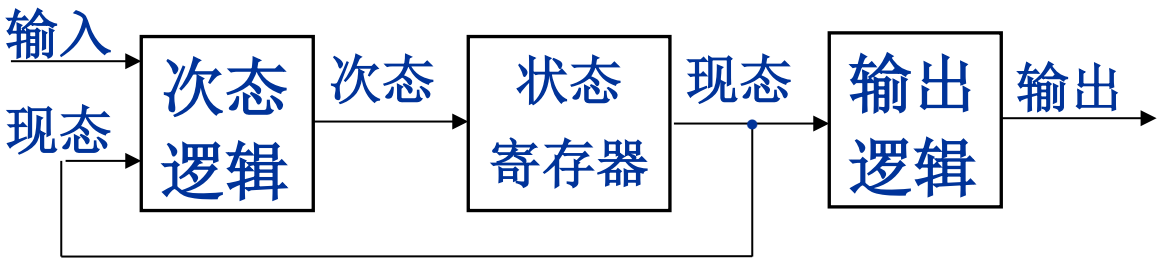
用n个宏名表示n个状态

```
｀define  state0=2｀b00 //不要加分号
｀define  state1=2｀b01
｀define  state2=2｀b11
｀define  state3=2｀b10
case (state)
｀  state0:……;
｀  state1:……;
   ……
```

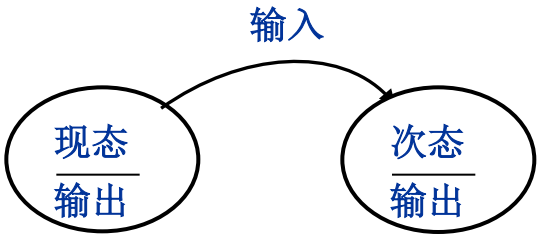
状态转换的描述

- 一般用case、casez或casex语句，比用if-else语句更清晰明了！
- 在case语句的最后，要加上default分支语句，以避免锁存器的产生。
- 状态机一般应设计为同步方式，并由一个时钟信号来触发。
- 实用的状态机都应设计为由唯一的时钟边沿触发的同步运行方式

Moore型有限状态机



Moore型状态机典型结构



Moore型状态图的表示

- Moore型状态机，其输出只为状态机当前状态的函数，而与外部输入无关。
- 外部输出是内部状态的函数。

Moore型有限状态机设计举例

【例2】设计一个序列检测器。要求检测器连续收到串行码{1101}后，输出检测标志为1，否则输出检测标志为0。

第1步：分析设计要求，列出全部可能状态：

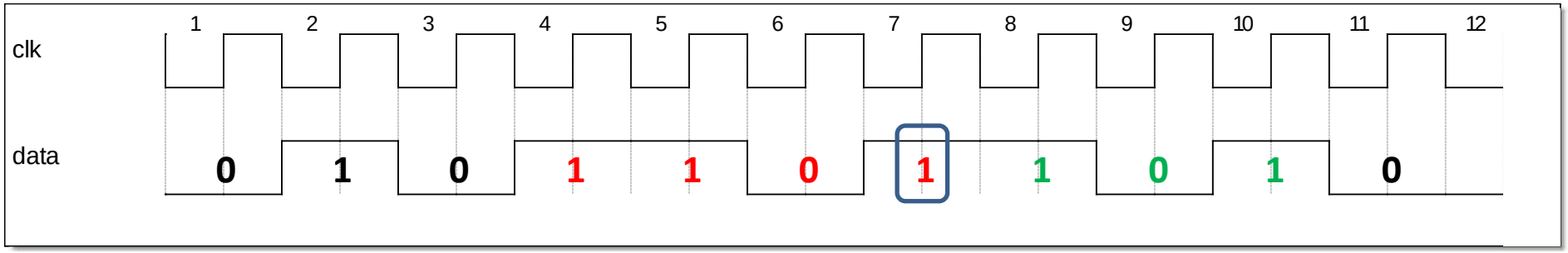
未收到一个有效位（0）	：S0
收到一个有效位（1）	：S1
连续收到两个有效位（11）	：S2
连续收到三个有效位（110）	：S3
连续收到四个有效位（1101）	：S4

- 由于序列检测器的输出只为状态机当前状态的函数，而与外部输入无关，所以为Moore型状态机

示例：序列检测器

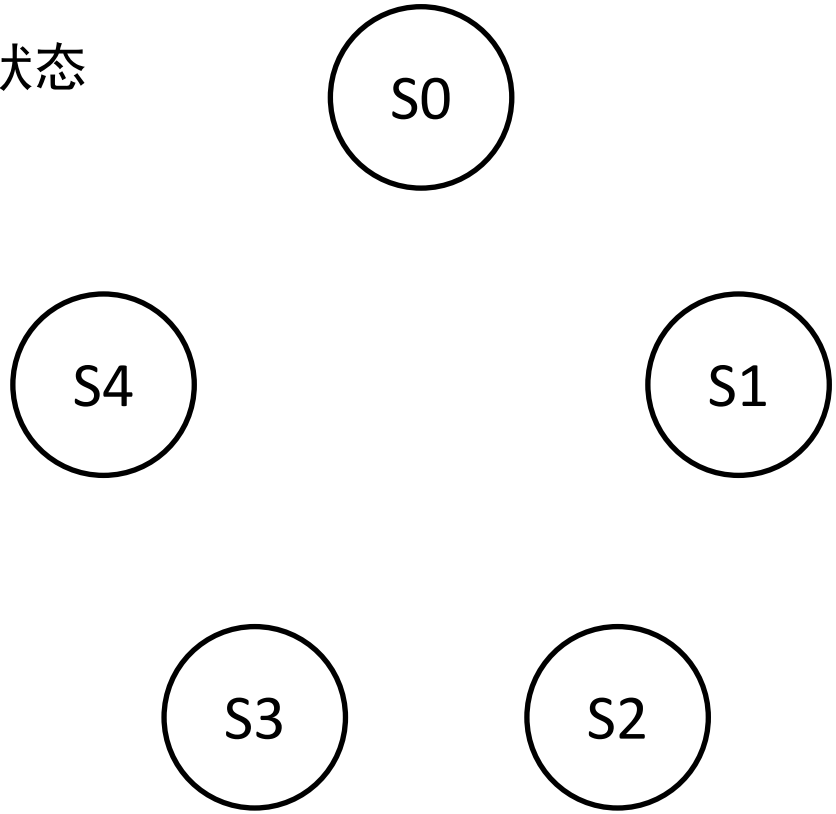
- 题目：设计一个序列检测器。要求检测器连续收到串行码{1101}后，输出检测标志为1，否则输出检测标志为0
- 题目修改：序列检测器的1位串行输入与时钟严格同步。当检测器连续收到串行码{1101}后，检测标志输出1且持续时间为1个cycle，否则输出0。
- 分析要点
  - 1）严格同步：意味着每个时钟周期，输入1位输入
  - 2）设计分歧：如何理解1101101？
    - 认为是2个{1101}或1个{1101}均可。倾向于2个。

Q： 如果data未与clk边沿对齐，是否影响设计？



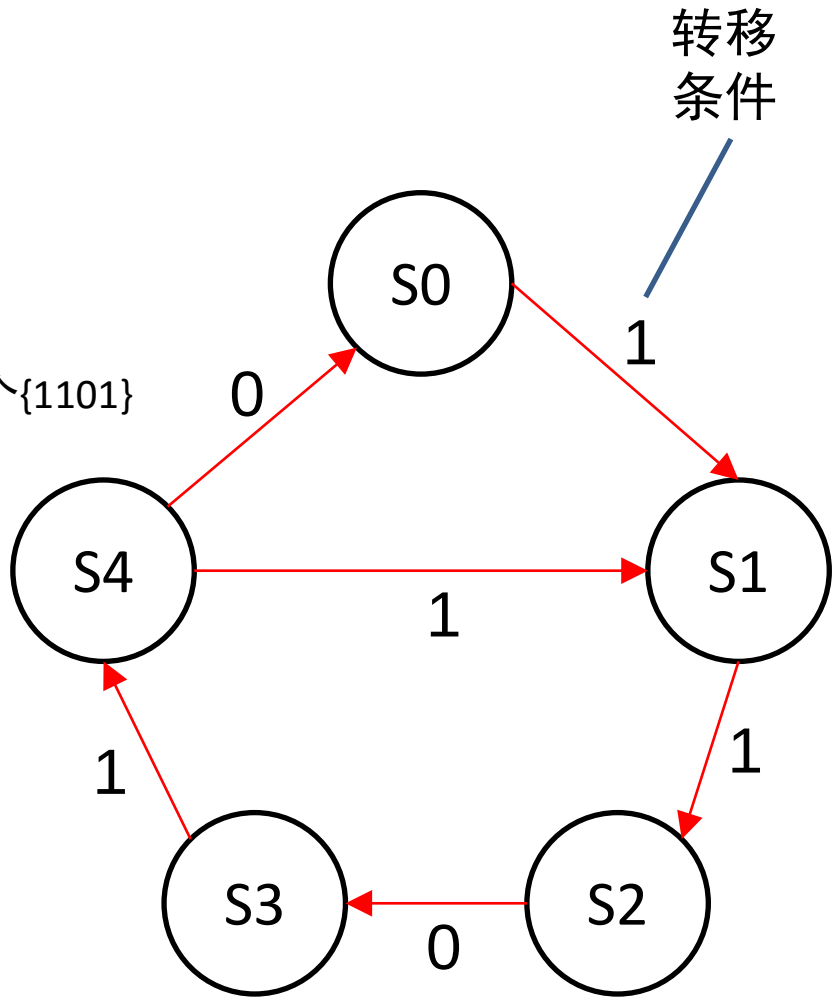
设计思路

- 基本原则：先定状态，后考虑输出
  - 现阶段不考虑MOORE或MEELY
- 每个时钟输入1位，可以用状态对应相应的匹配阶段
- STEP1：状态规划
  - 4个cycle才能完全输入，所以需要4个状态
    - S1：匹配{1}
    - S2：匹配{11}
    - S3：匹配{110}
    - S4：匹配{1101}
  - S0：无匹配成功



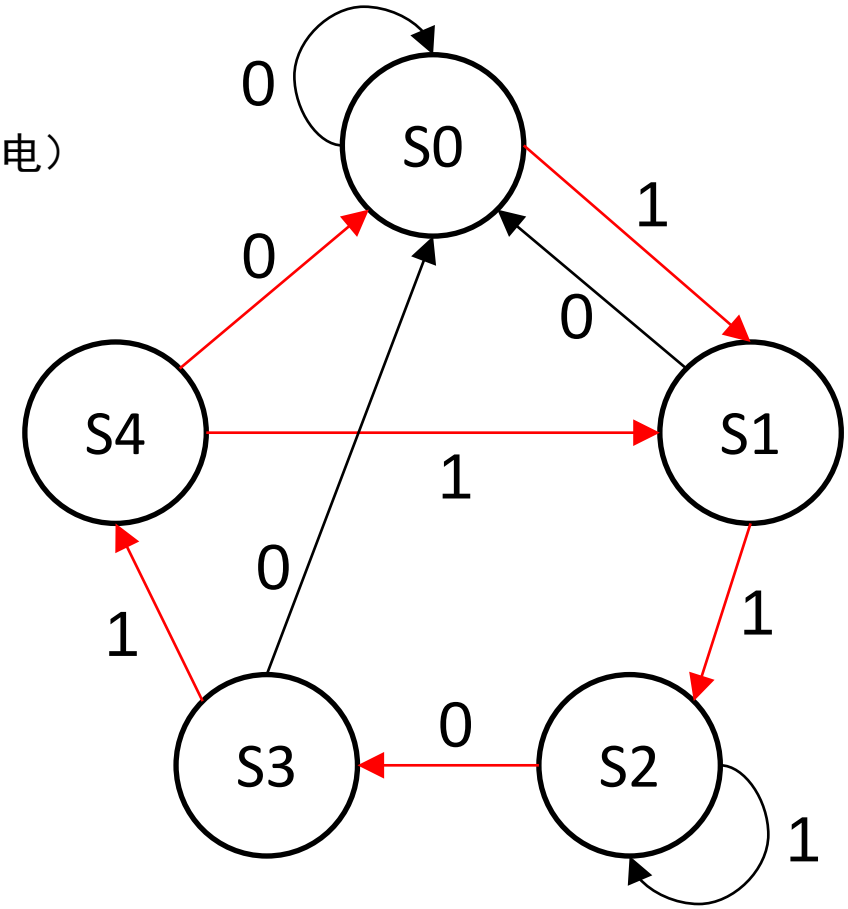
设计思路

- STEP2：确定状态转换
  - 一般需要选择一个初始状态作为起始状态
    - S0：在本例中是最好的初始状态
  - 先确定状态迁移的核心路径（红色）
    - 在状态迁移箭头上标记转移条件
    - S4：体现了前述的设计分歧
      - 当前设计不能识别{1101101}中的第2个{1101}



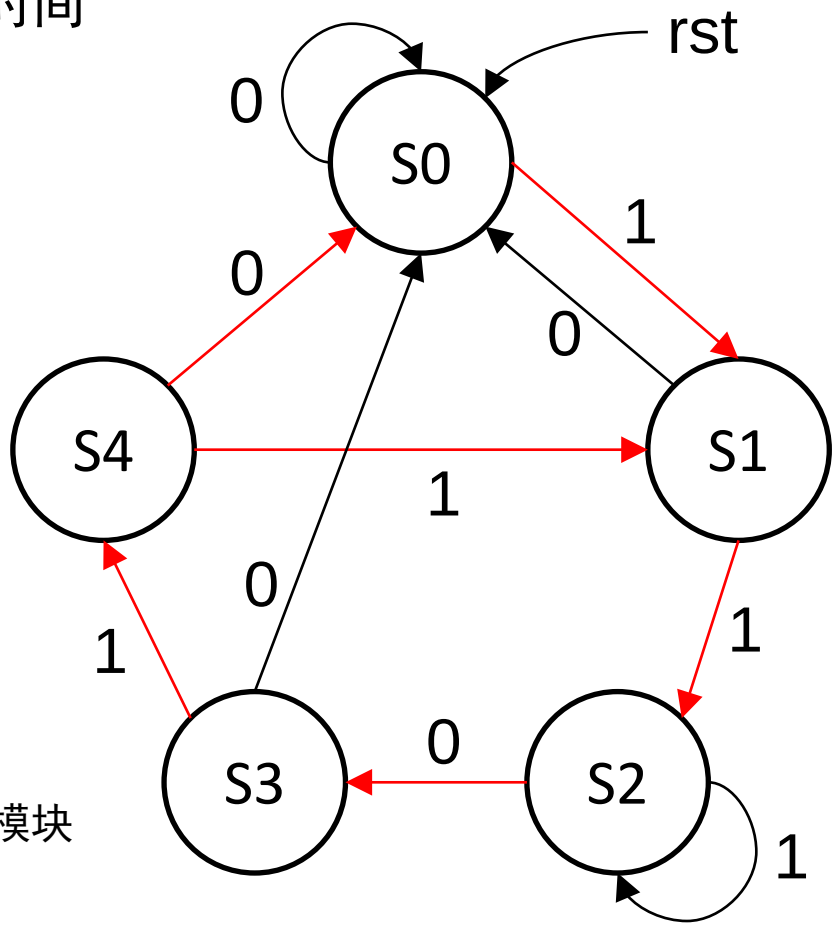
设计思路

- STEP3：处理所有其他可能
  - 避免极端状态：只进不出 or 只出不进
    - 只进不出：结束状态
      - 一般用于彻底停机
    - 只出不进：极其原始的状态
      - 一般用于对整个电路进行复位（刚上电）



设计思路

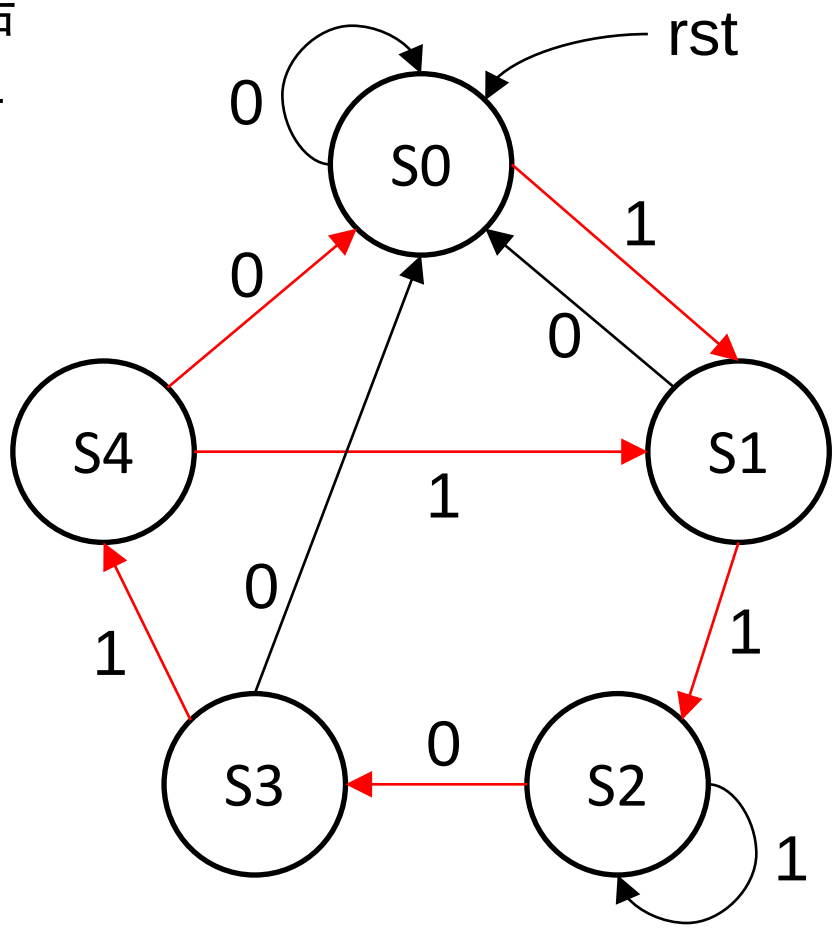
- STEP4：增加复位（reset）信号
  - 多数数字电路设计都会考虑复位功能
    - 确保所有数字电路在初始化阶段能被有效初始化，从而全系统能严格同步
  - 初始化阶段：大多数为上电后的短暂时间
    - 一般以ms为单位
  - Reset产生机制
    - 某个系统级部件产生 or
    - 某个控制器可复位其他控制器
  - Reset电平规划
    - 高电平H(1) or 低电平L(0)均可
    - 芯片外部输入：一般为L有效
    - 芯片内部使用：一般用H有效
      - 顶层输入后做反相变换再传输给其他模块



设计思路

- STEP5：产生输出信号
  - 设计分歧：Moore？Meely？
  - 决策机制：取决于期望输出信号在什么时候有效！
  - Moore最早有效：状态机进入S4状态后
  - Meely最早有效：状态机进入S3状态后
    - 注意：输入信号也是判断条件之一

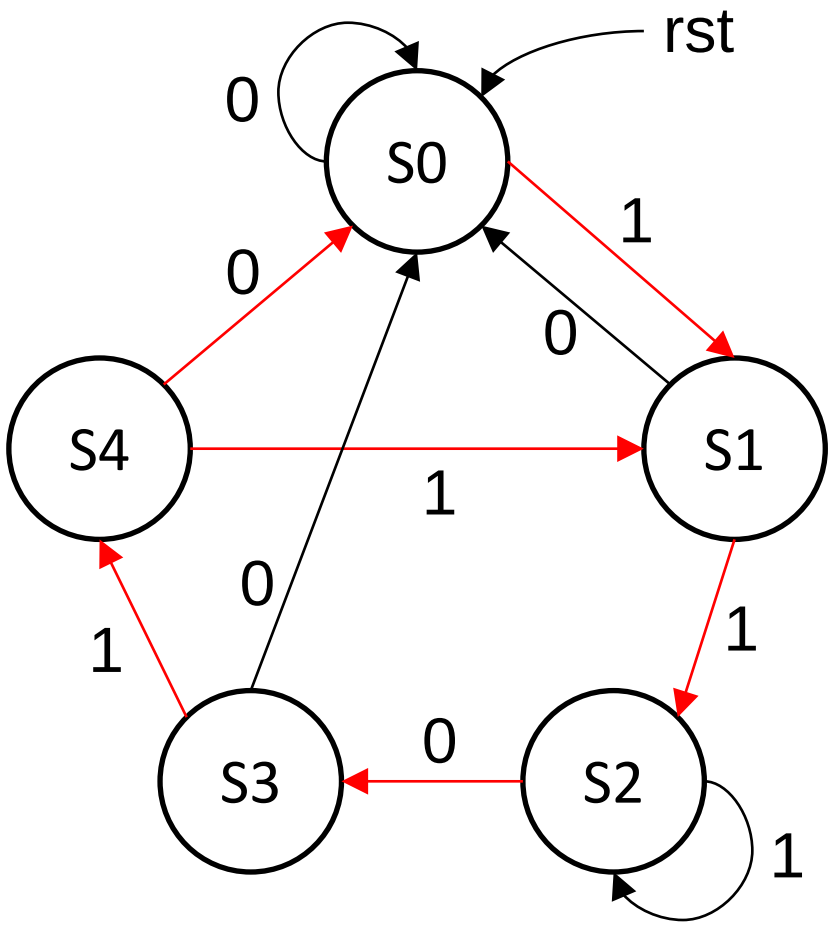
```
Moore:
zo = (fsm==S4)

Meely:
zo = (fsm==S3) & data
```



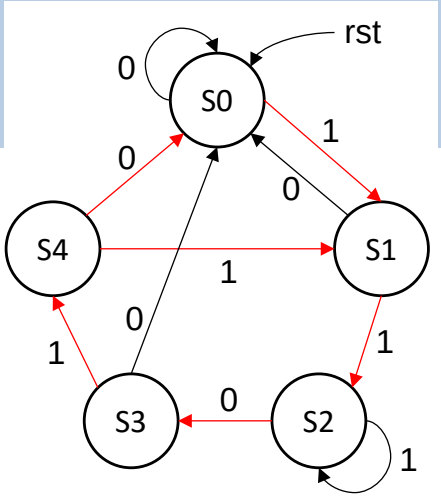
设计思路

- STEP6：构造次态逻辑真值表；生成寄存器的次态逻辑表达式
  - 假设编码如下：{S0：000，S1：001，S2：010，S3：011，S4：100}

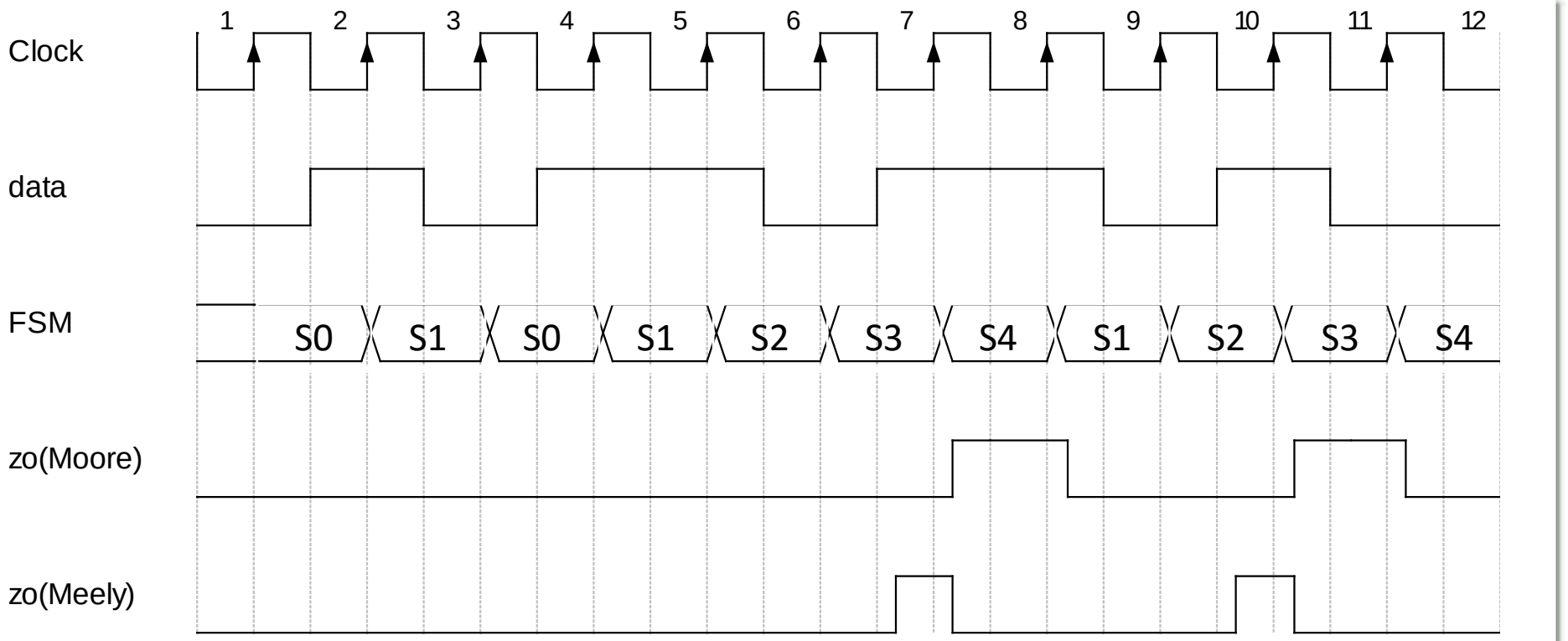
输入	现态	次态
0	000	000
1	000	001
0	001	000
1	001	010
0	010	011
1	010	010
0	011	000
1	011	100
0	100	000
1	100	001
X	101	000
X	110	000
X	111	000



设计思路

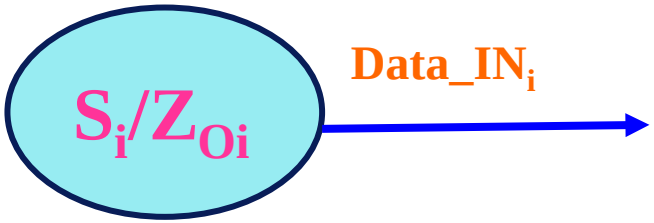


- 时序分析
  - Moore：输出晚，有效时间以cycle为单位
  - Meely：可以很早，有效时间通常不以cycle为单位

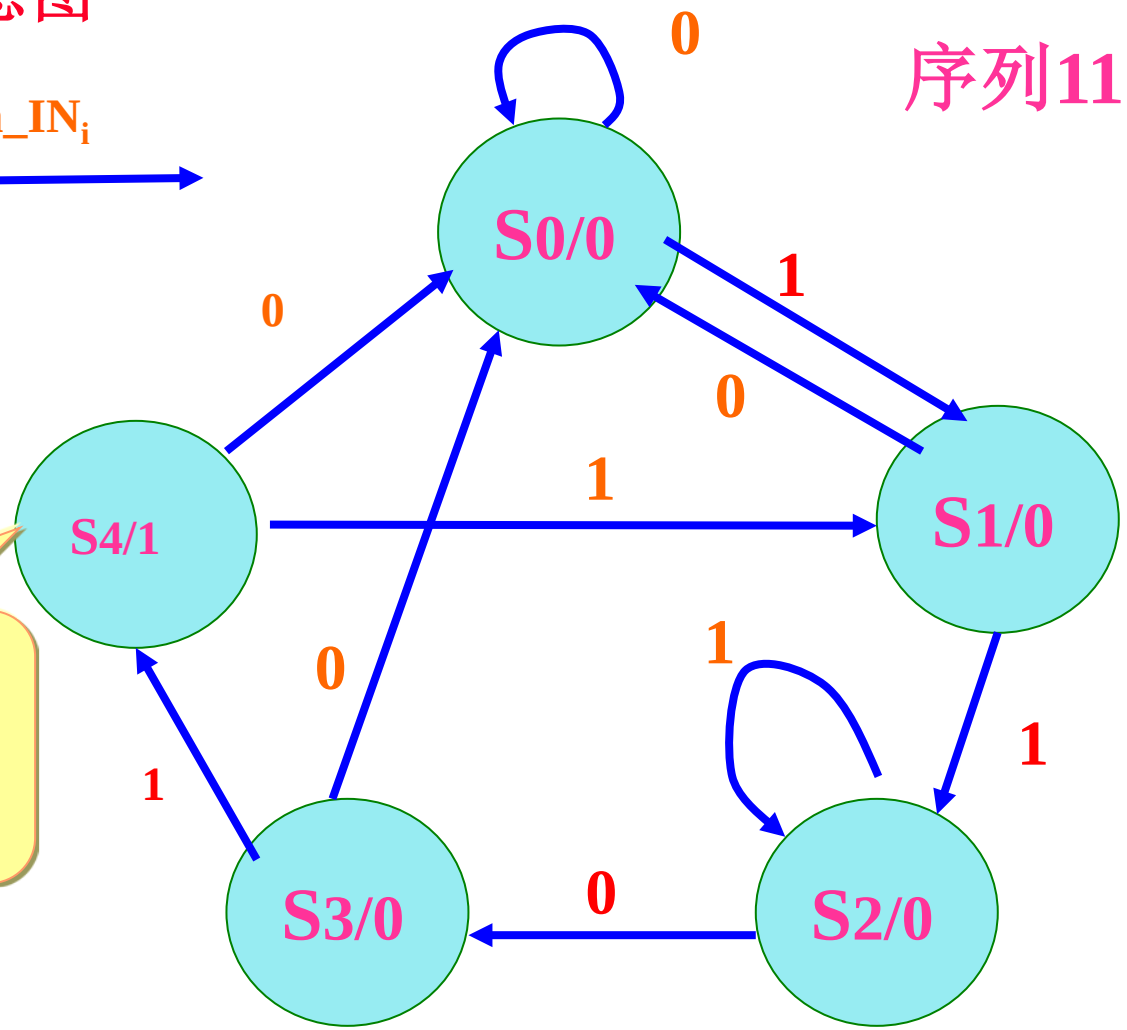


Moore型有限状态机设计举例

第2步：画出状态图



序列1101



只有在S4状态下输出zo置1，其他状态下zo为0，且输出与外部输入无关

Moore型有限状态机设计举例

第3步：用Verilog语言描述状态机

- 在程序的开头定义状态机状态的编码形式
  - 用**parameter**或｀**define**语句
- 复位时回到起始状态
  - 敏感信号为时钟和复位信号
- 状态转换描述
  - 用**case**或**if-else**语句描述出状态的转移（根据现态和输入产生次态，可与复位时回到起始状态的语句放在同一个**always**块中，即敏感信号为时钟和复位信号）
- 输出信号描述
  - 用**case**语句描述状态机的输出信号（单独放在一个**always**块中，敏感信号为现态）

序列检测器源程序（Moore型状态机）

```
module monitor(clk,clr,data,zo,state);
   parameter S0=3'b000, S1=3'b001,
   S2=3'b010,S3=3'b011,S4=3'b100; //状态编码的定义
   input clk,clr,data;
   output zo;
   output[2:0] state;                                    //状态机
   reg [2:0] state;
   reg zo;
   always @(posedge clk or posedge clr)
     begin
        if (clr) state=S0; //（1）复位时回到初始状态
        else
          begin
            case (state)//  （2）状态的转移
              S0: if (data==1'b1) state=S1;
                     else state=S0;
               S1: if (data==1'b1) state=S2;
                     else state=S0;
              S2: if (data==1'b0) state=S3;
                     else state=S2;
               S3: if (data==1'b1) state=S4;
                    else state=S0;
              S4: if (data==1'b1) state=S1;
                     else state=S0;
               default: state=S0;
            endcase
            zo=(state==S4)?1'b1:1'b0;  //（3）状态机的输出信号
          end
     end
endmodule
```

在S4时给zo置1——输出只为状态机当前状态的函数，而与外部输入无关

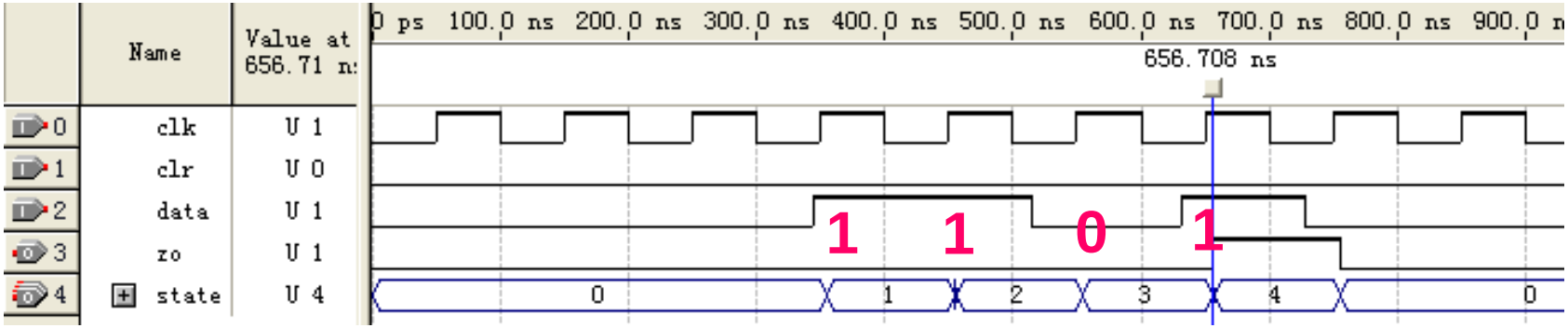
状态机的输出信号描述

- 如果输出表达式很简单，可以单独用一条赋值语句写出
- 最好将状态机的输出语句单独写在一个always块中，这样逻辑清晰，不易出错：

语句“zo=(state==S4)?1’b1:1’b0;”去掉，换成如下always块（组合逻辑）

```
always @(state)      //（3）状态机的输出信号
   begin
      case (state)
         S0:  zo=1'b0;
         S1:  zo=1'b0;
         S2:  zo=1'b0;
         S3:  zo=1'b0;
         S4:  zo=1'b1;
         default: zo=1'b0;
      endcase
   end
```

序列检测器的仿真波形



- 状态机的Verilog描述一般采用双过程描述：一个过程描述现态和次态时序逻辑；另一个过程描述输出逻辑（组合逻辑）。这样写结构清晰；将时序逻辑和组合逻辑分开描述，便于修改。
- 描述包括3个部分：

  （1）复位时回到初始状态；

  （2）状态的转移；

  （3）状态机的输出信号。

第三部分：时序逻辑电路设计

一. 锁存器和触发器

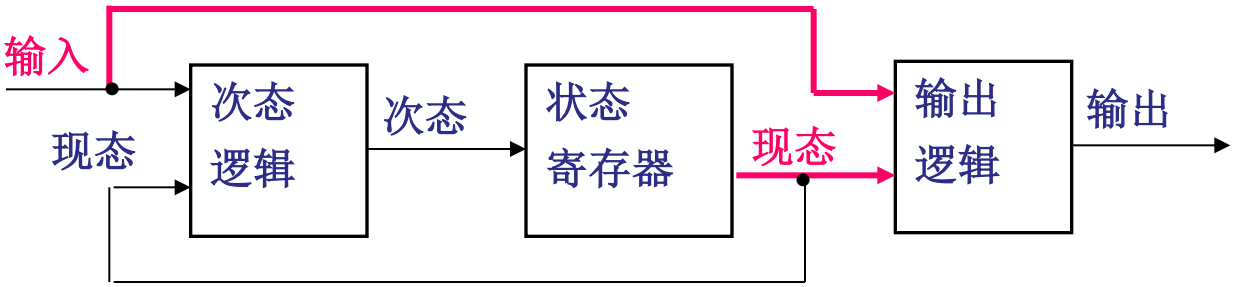
二. 有限状态机

1. Moore型有限状态机

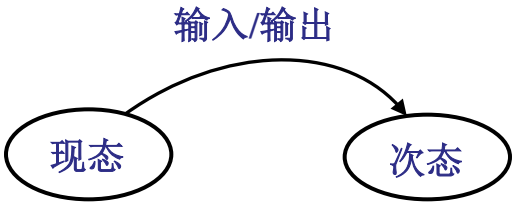
2. Mealy型有限状态机

三. 时序逻辑电路设计分析

Mealy型有限状态机



Mealy型状态机的典型结构

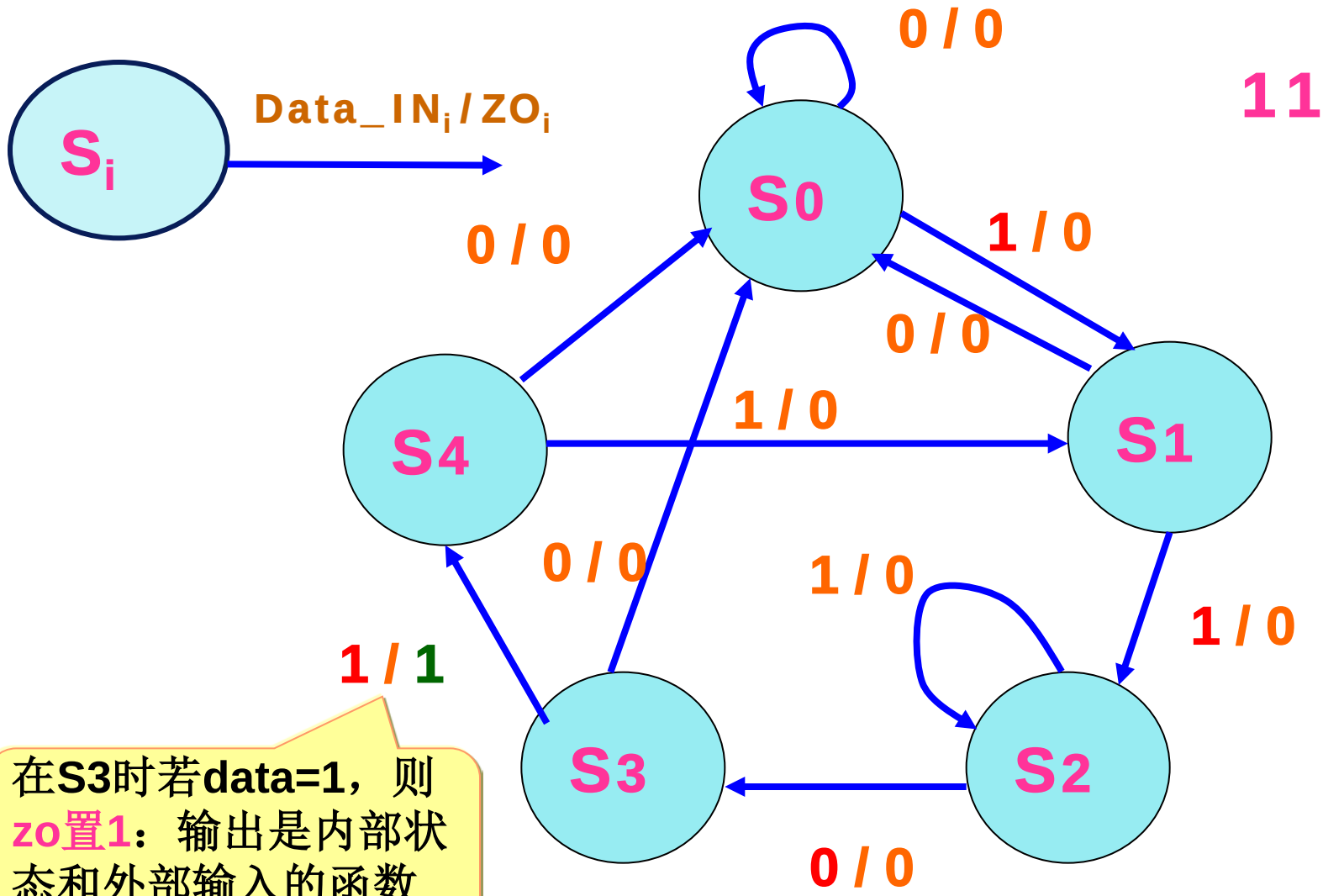


Mealy型状态图的表示

- Mealy型状态机，其输出不仅与状态机当前状态有关，而且与输入有关——外部输出是内部状态和外部输入的函数。
- 在“Mealy型状态图的表示”中，每个圆圈表示状态机的一个状态，每个箭头表示状态之间的一次转移；引起状态转换的输入信号及产生的输出信号标注在箭头上。

【例3】将【例2】采用Mealy型状态机实现。

序列检测器的Mealy型有限状态机状态图



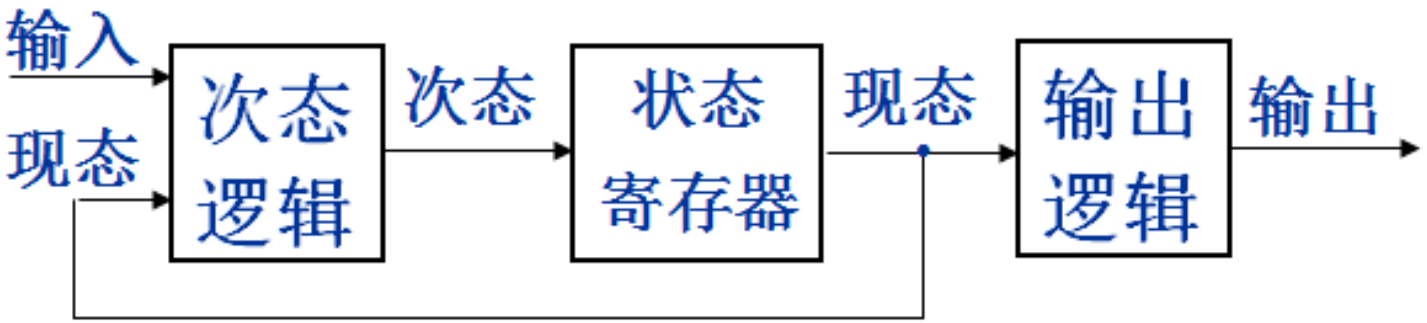
在S3时若data=1，则zo置1：输出是内部状态和外部输入的函数

序列检测器源程序（Mealy型状态机）

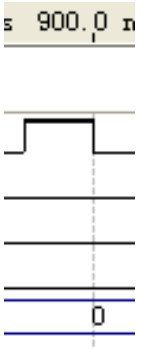
```
module monitor2_good(clk,clr,data,zo,state);
   parameter S0=3'b000, S1=3'b001,
   S2=3'b010,S3=3'b011,S4=3'b100;
   input clk,clr,data;
   output zo;
   output[2:0] state;
   reg [2:0] state;
   reg zo;
   always @(posedge clk or posedge clr)
     begin
        if (clr) state=S0; //（1）复位时回到初始状态
        else
          begin
             case (state) //（2）状态的转移
                S0:   if (data==1'b1) state=S1;
                      else state=S0;
                S1:   if (data==1'b1) state=S2;
                      else state=S0;
                S2:   if (data==1'b0) state=S3;
                      else state=S2;
                S3:  if (data==1'b1) state=S4;
                      else state=S0;
                S4:  if (data==1'b1) state=S1;
                      else state=S0;
                default: state=S0;
             endcase
          end
     end
endmodule
```

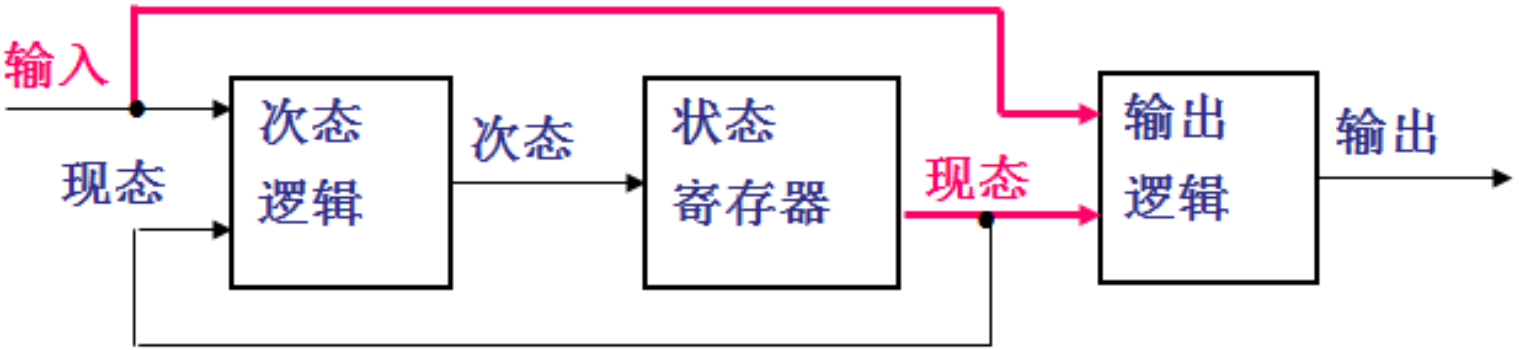
序列检测器源程序及仿真波形

```
always @(state)      //（3）状态机的输出信号
  begin
    case (state)
      S0: z=1'b0;
```



Moore型状态机典型结构





Mealy型状态机的典型结构